МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет

имени академика С.П. Королева» (Самарский университет)

Институт информатики и кибернетики Кафедра технической кибернетики

**ОТЧЕТ по ЛР 2-3**

**«Разработка API & Авторизация»**

Выполнили студенты группы 6303-010302D

Щетинин Михаил Маргарян Гор

САМАРА 2025

1. О токенах в проекте

Для нашего сайта мы реализовали access token, имеющий срок годности в 24 часа с его выдачи. По истечению своего срока пользователю придется повторно осуществить вход для получения доступа к информации для авторизованных игроков. При это истекший токен будет занесен в таблицу Blacklistedtoken, имеющей следующую структуру:

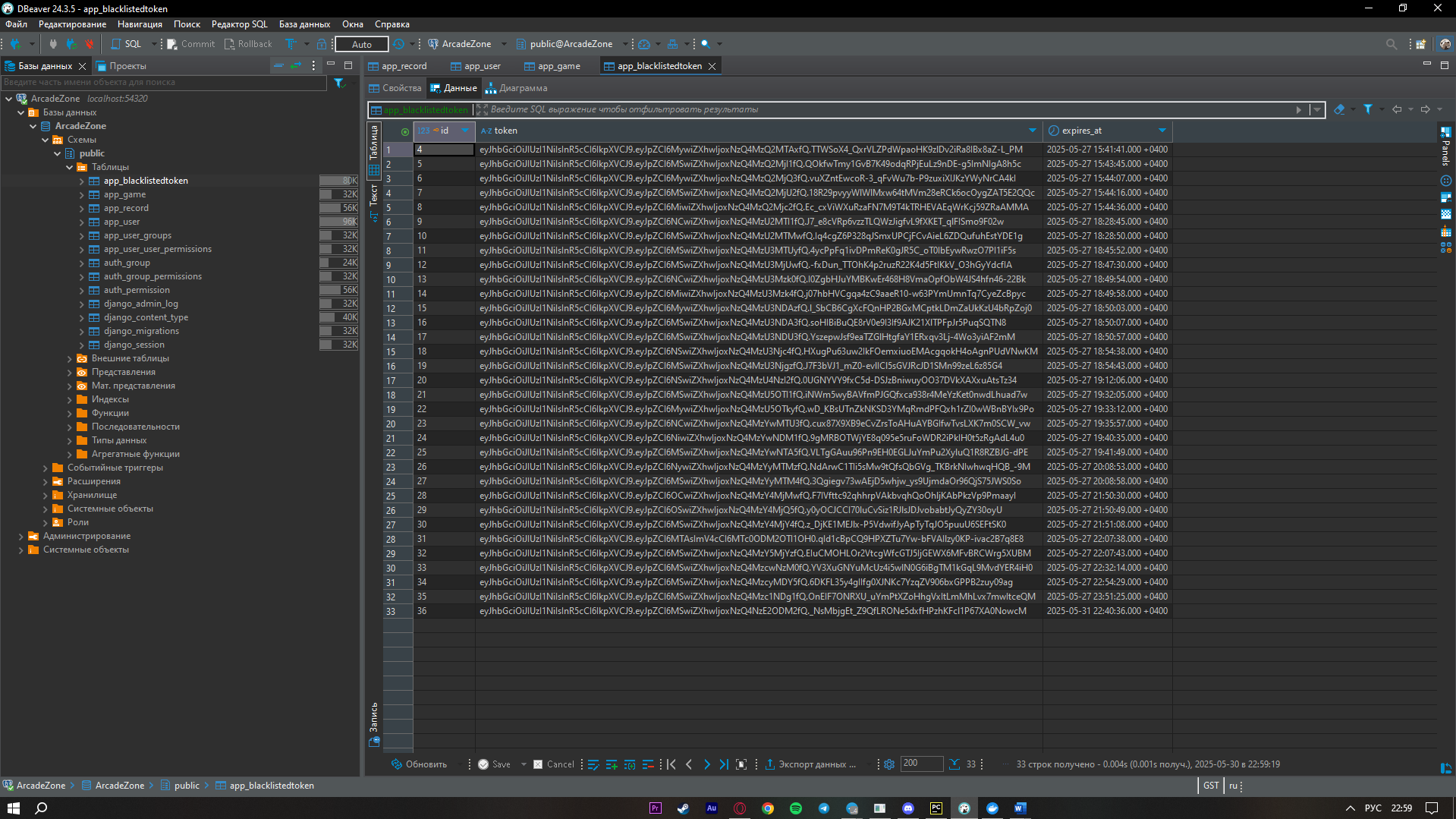


Рисунок 1 – Таблица BlacklistedToken

1. Тестирование методов аутентификации и кодов ошибок

Тестирование методов выполнялось в ПО Postman.

Для метода регистрации игрока свойственны http-коды:

* 201, если пользователь успешно зарегистрировался

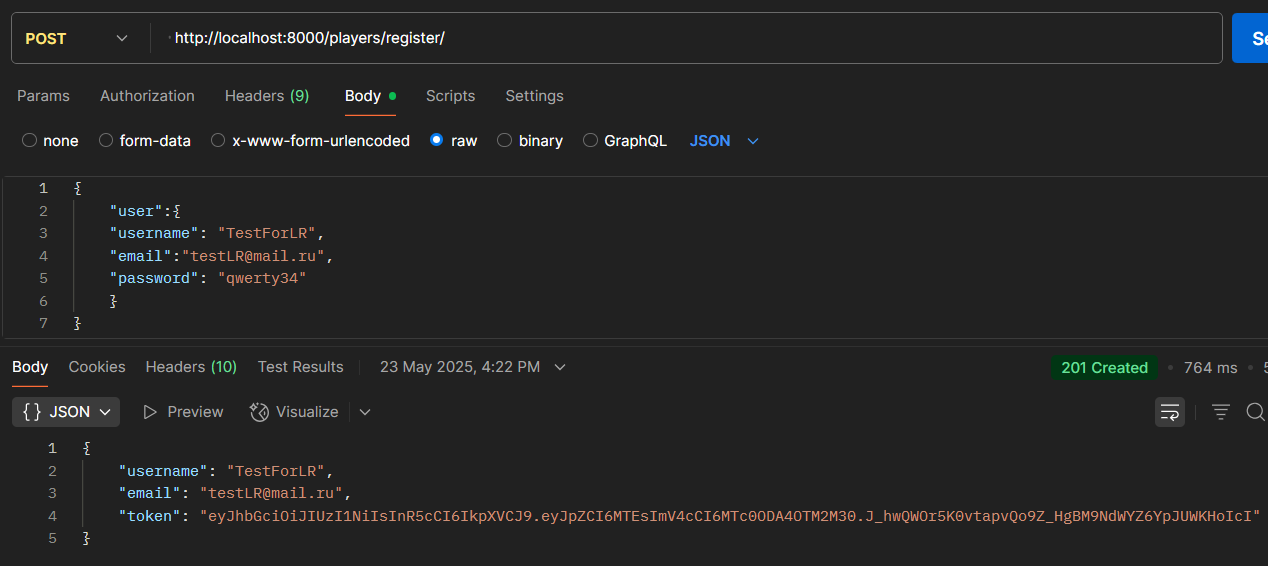


Рисунок 2 – Игрок успешно зарегистрировался

* 400, если тело запроса имеет некорректные данные

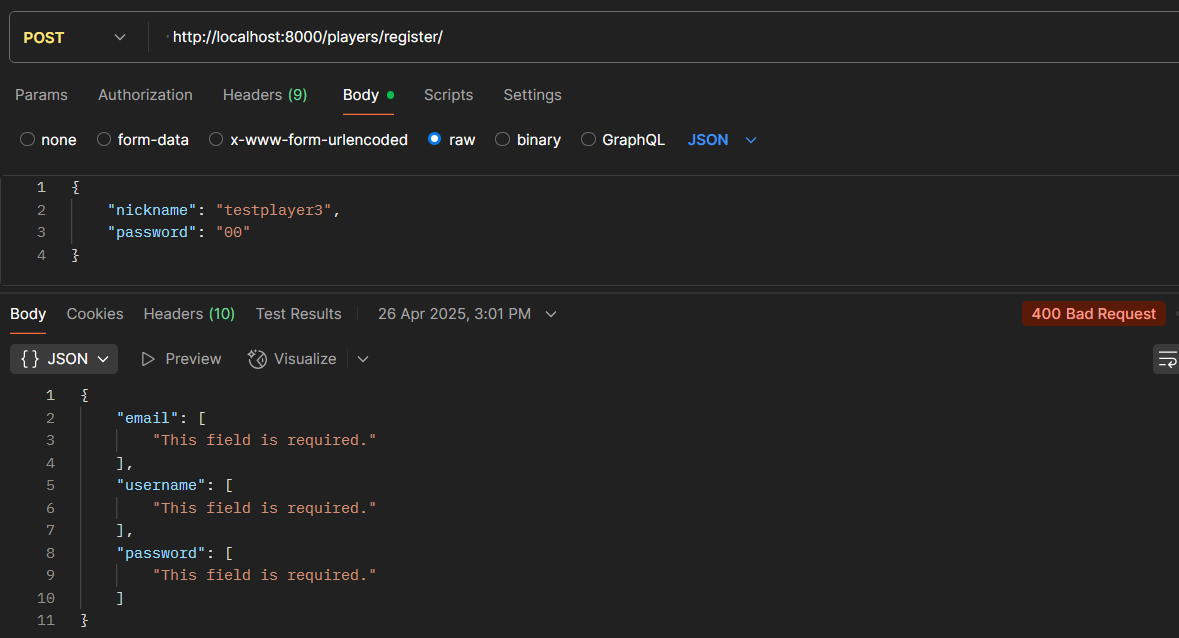


Рисунок 3 – Игрок не смог зарегистрироваться

Для метода login\_player свойственны следующие http-коды:

* 200, если пользователь успешно вошел в систему

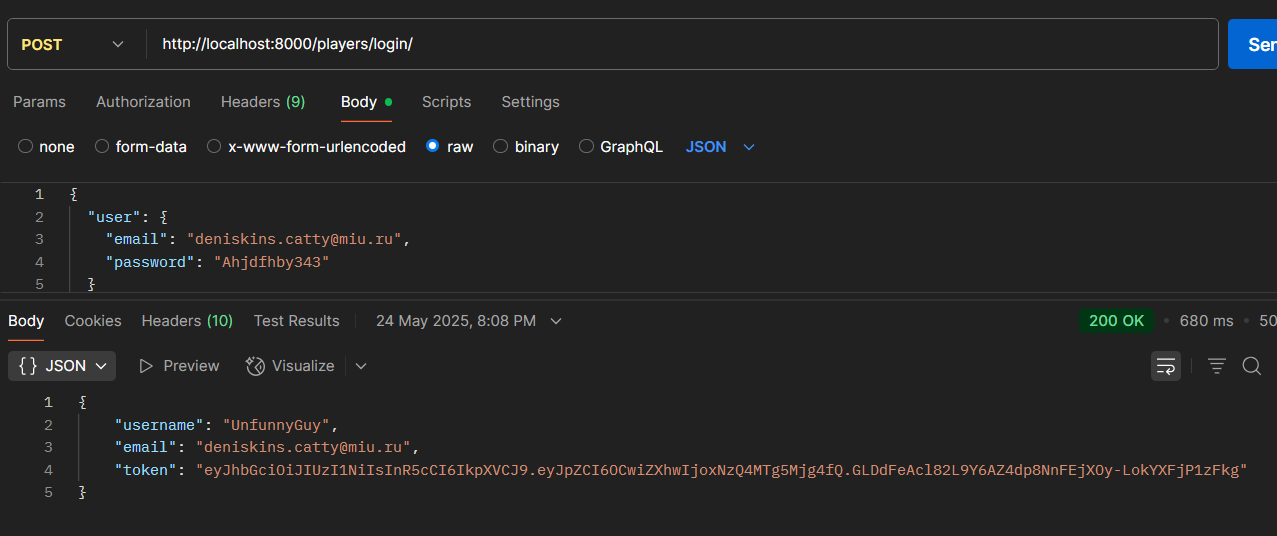


Рисунок 4 – Пользователь успешно зашел на сайт

* 400, если тело запроса содержит ошибки

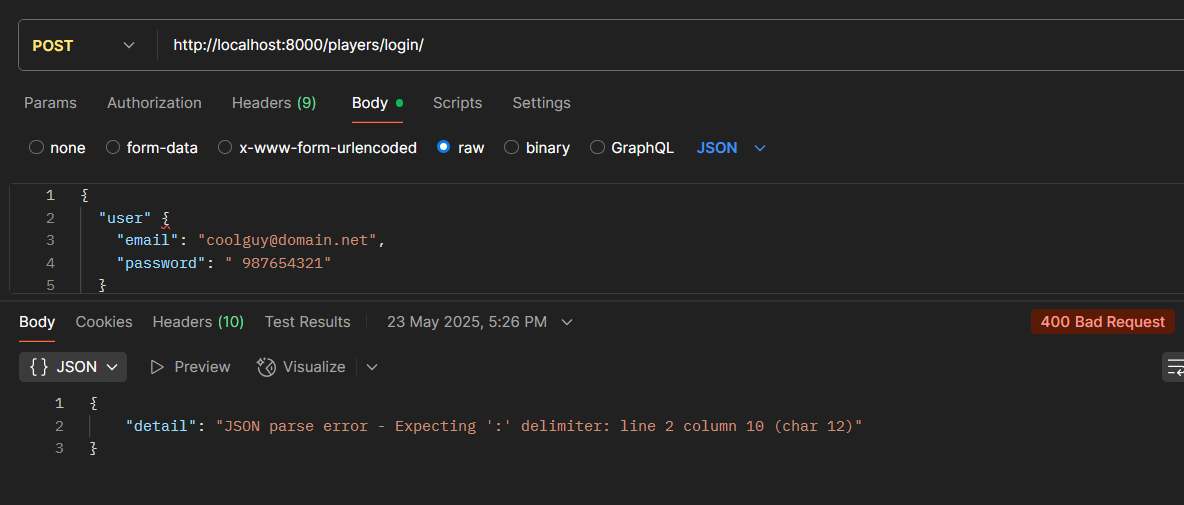


Рисунок 5 – Пользователь не смог войти

Для метода logout\_player свойственны следующие http-коды:

* 200, если человек вышел из системы

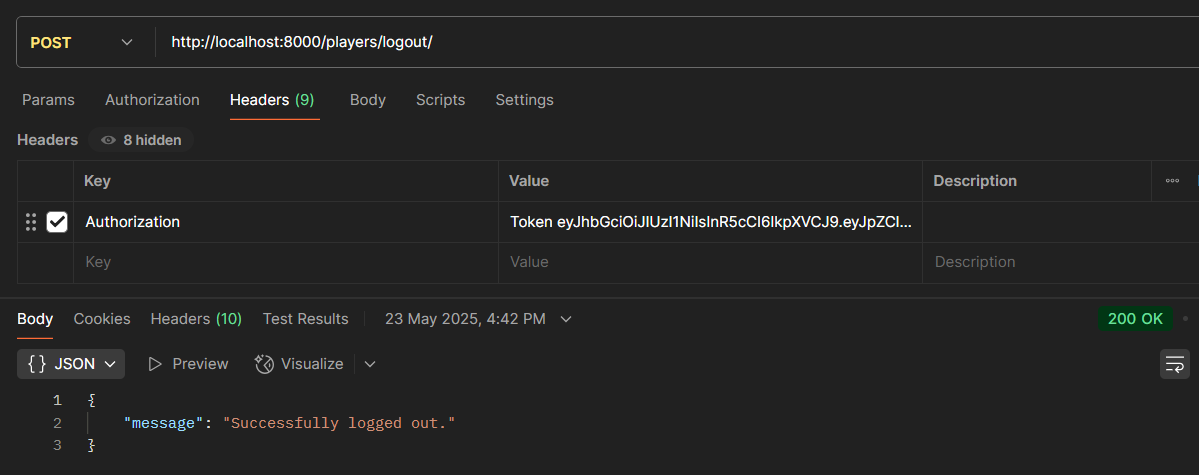


Рисунок 6 – человек вышел из системы

* 403, если человек пытается выйти из сайта с истекшим токеном

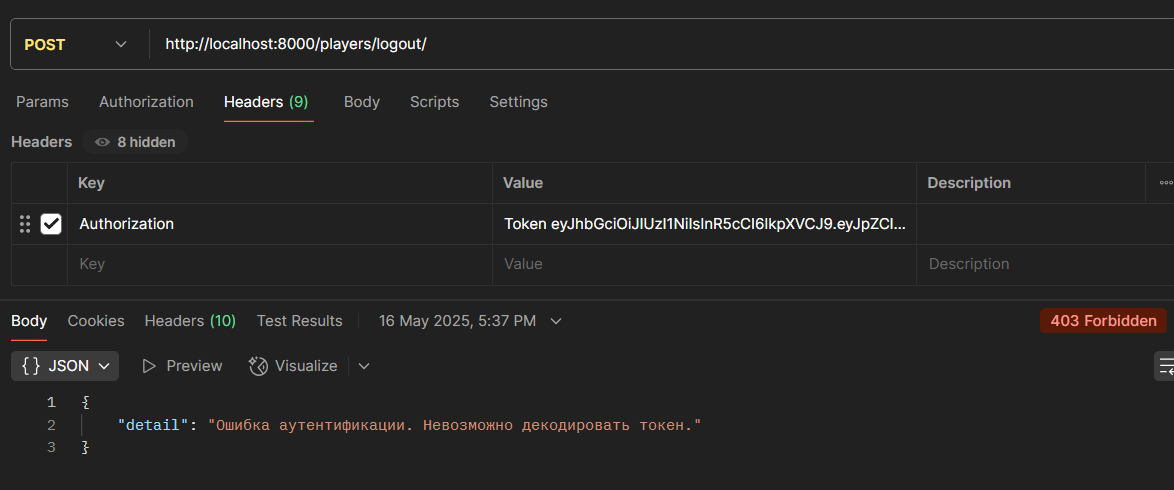


Рисунок 7 – Человек не смог выйти из системы

Для метода get\_game свойственны следующие http-коды:

* 200, если игрок получил игру

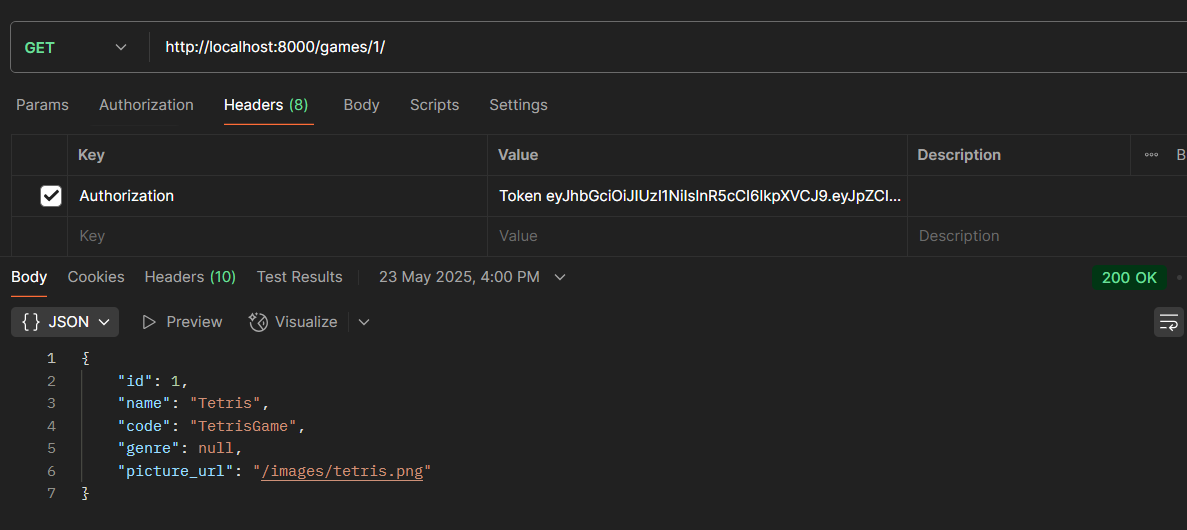


Рисунок 8 – Игрок получил игру

* 403, если у него токена нет или токен не прошел валидацию

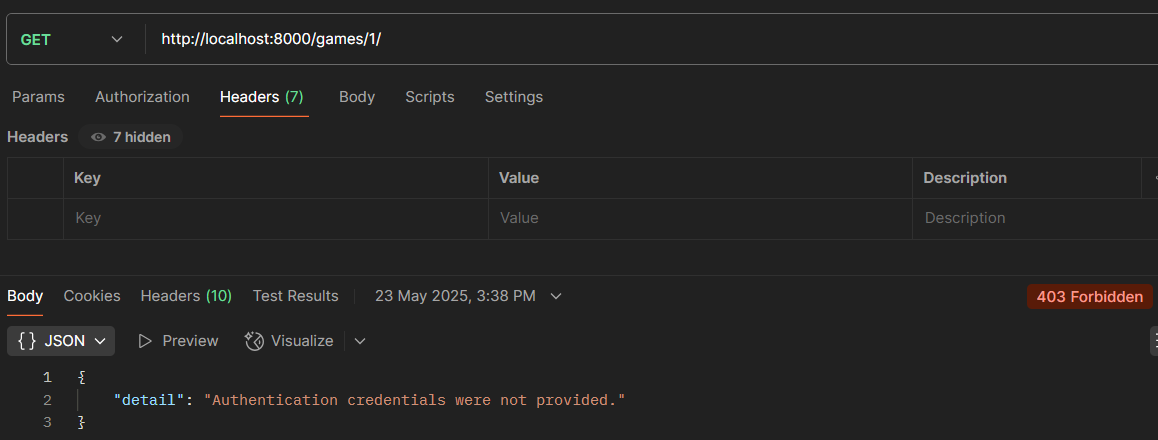


Рисунок 9 – Пользователь не смог получить игру

Для метода add\_game свойственны http-коды:

* 200, если удалось добавить игру

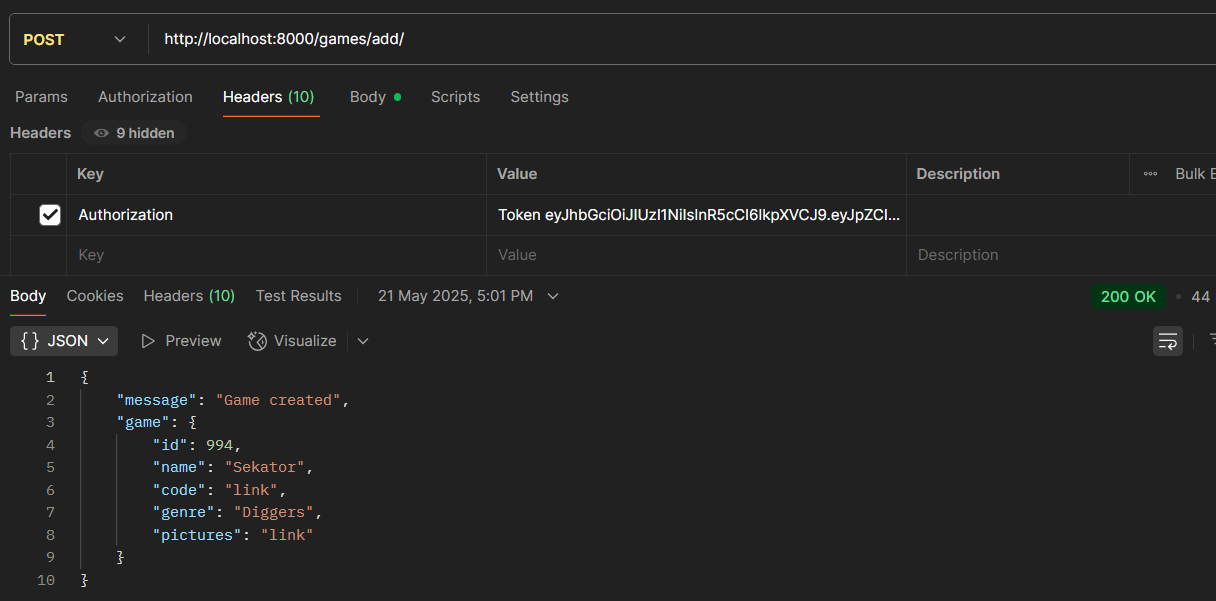


Рисунок 9 – Админу удалось добавить игру

* 403, если токен админа не прошел валидацию

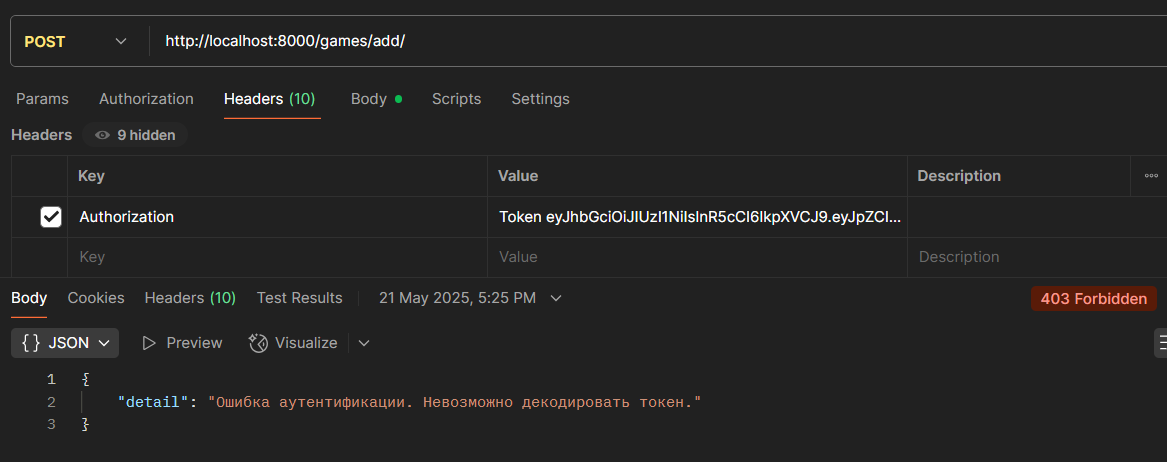


Рисунок 10 – Админ не смог добавить игру

Стоит также отметить, что на ранних этапах реализации и в случае неграмотной поддержки сайта, каждый из упомянутых методов может выдавать код 500 – Internal Server Error.

Приложение А

#Код сериализаторов, вызываемых CRUD – методами.

from django.utils import timezone

from datetime import timezone as dt\_timezone

from django.conf import settings

import jwt

from django.utils.timezone import now

from rest\_framework import serializers

from django.contrib.auth.hashers import make\_password

from .models import User, BlackListedToken

from django.contrib.auth import authenticate

from .clear\_expired\_tokens import Command

class RegistrationSerializer(serializers.ModelSerializer):

    # Убедитесь, что пароль содержит не менее 8 символов, не более 128,

    # и так же что он не может быть прочитан клиентской стороной

    password = serializers.CharField(

        max\_length=128,

        min\_length=8,

        write\_only=True

    )

    # Клиентская сторона не должна иметь возможность отправлять токен вместе с

    # запросом на регистрацию. Сделаем его доступным только на чтение.

    token = serializers.CharField(max\_length=255, read\_only=True)

    class Meta:

        model = User

        # Перечислить все поля, которые могут быть включены в запрос

        # или ответ, включая поля, явно указанные выше.

        fields = ['email', 'username', 'password', 'token']

    def create(self, validated\_data):

        user = User.objects.create\_user(\*\*validated\_data)

        user.last\_login = now()

        user.save(update\_fields=['last\_login'])

        return user

class LoginSerializer(serializers.Serializer):

    email = serializers.CharField(max\_length=255)

    username = serializers.CharField(max\_length=255, read\_only=True)

    password = serializers.CharField(max\_length=128, write\_only=True)

    token = serializers.CharField(max\_length=255, read\_only=True)

    def validate(self, data):

        # В методе validate мы убеждаемся, что текущий экземпляр

        # LoginSerializer значение valid. В случае входа пользователя в систему

        # это означает подтверждение того, что присутствуют адрес электронной

        # почты и то, что эта комбинация соответствует одному из пользователей.

        email = data.get('email', None)

        password = data.get('password', None)

        # Вызвать исключение, если не предоставлена почта.

        if email is None:

            raise serializers.ValidationError(

                'Требуется email'

            )

        # Вызвать исключение, если не предоставлен пароль.

        if password is None:

            raise serializers.ValidationError(

                'Требуется пароль'

            )

        # Метод authenticate предоставляется Django и выполняет проверку, что

        # предоставленные почта и пароль соответствуют какому-то пользователю в

        # нашей базе данных. Мы передаем email как username, так как в модели

        # пользователя USERNAME\_FIELD = email.

        user = authenticate(username=email, password=password)

        # Если пользователь с данными почтой/паролем не найден, то authenticate

        # вернет None. Возбудить исключение в таком случае.

        if user is None:

            raise serializers.ValidationError(

                'Пользователь с таким паролем и email не был найден'

            )

        # Django предоставляет флаг is\_active для модели User. Его цель

        # сообщить, был ли пользователь деактивирован или заблокирован.

        # Проверить стоит, вызвать исключение в случае True.

        if not user.is\_active:

            raise serializers.ValidationError(

                'Этот пользователь деактивирован'

            )

        user.last\_login = now()

        user.save(update\_fields=['last\_login'])

        # Метод validate должен возвращать словарь проверенных данных. Это

        # данные, которые передаются в т.ч. в методы create и update.

        return {

            'email': user.email,

            'username': user.username,

            'token': user.token

        }

class UserSerializer(serializers.ModelSerializer):

    # Осуществляет сериализацию и десериализацию объектов User.

    # Пароль должен содержать от 8 до 128 символов. Это стандартное правило. Мы

    # могли бы переопределить это по-своему, но это создаст лишнюю работу для

    # нас, не добавляя реальных преимуществ, потому оставим все как есть.

    email = serializers.EmailField(required=False, allow\_blank=True)

    username = serializers.CharField(required=False, allow\_blank=True)

    password = serializers.CharField(

        max\_length=128,

        min\_length=8,

        write\_only=True,

        required=False,

        allow\_blank=True

    )

    current\_password = serializers.CharField(

        write\_only=True,

        required=True,

    )

    class Meta:

        model = User

        fields = ('email',

                  'username',

                  'password',

                  'current\_password',

                  'token',

                  'created\_at',

                  'updated\_at',

                  'is\_staff',

                  'is\_superuser',

                  )

        read\_only\_fields = ('token',)

    def validate\_current\_password(self, value):

        user = self.instance

        if not user.check\_password(value):

            raise serializers.ValidationError('Неверный текущий пароль.')

        return value

    def update(self, instance, validated\_data):

        # Выполняет обновление User.

        validated\_data.pop('current\_password', None)  # Проверка старого пароля

        password = validated\_data.pop('password', None)  # Новый пароль (если имеется)

        email = validated\_data.get('email')

        if email == '':

            validated\_data.pop('email')

        username = validated\_data.get('username')

        if username == '':

            validated\_data.pop('username')

        for key, value in validated\_data.items():

            setattr(instance, key, value)

        if password is not None and password != '':

            instance.set\_password(password)

        instance.save()

        return instance

    def logout(self):

        # Выход пользователя: занести токен в чёрный список.

        request = self.context.get('request')

        if request is None:

            raise serializers.ValidationError('Request context is required.')

        # Достаём токен из заголовков

        auth\_header = request.META.get('HTTP\_AUTHORIZATION')

        if not auth\_header:

            raise serializers.ValidationError('Authorization header missing.')

        # Пример: Authorization: Token <token>

        try:

            token\_str = auth\_header.split()[1]

        except IndexError:

            raise serializers.ValidationError('Token missing in header.')

        # Заносим токен в Blacklist

        com = Command()

        com.update\_blacklist(token\_str)

#Код функций по аутентификации, вызывающих эти сериализаторы

@api\_view(['POST'])

@permission\_classes([AllowAny])

def register\_player(request):

    user = request.data.get('user', {})

    serializer = RegistrationSerializer(data=user)

    if serializer.is\_valid():

        serializer.save()

        return Response({

            "username": serializer.data["username"],

            "email": serializer.data["email"],

            "token": serializer.data["token"]

        }, status=status.HTTP\_201\_CREATED)

    return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['POST'])

@permission\_classes([AllowAny])

def login\_player(request):

    user = request.data.get('user', {})

    serializer = LoginSerializer(data=user)

    serializer.is\_valid(raise\_exception=True)

    return Response({

        "username": serializer.data["username"],

        "email": serializer.data["email"],

        "token": serializer.data["token"]

    }, status=status.HTTP\_200\_OK)

@api\_view(['POST'])

@permission\_classes([IsAuthenticated])

def logout\_player(request):

    # Выход из аккаунта. Добавляем токен в чёрный список.

    serializer = UserSerializer(

        request.user,

        context={'request': request}  # Чтобы передать request внутрь сериализатора

    )

    serializer.logout()

    return Response({'message': 'Successfully logged out.'}, status=status.HTTP\_200\_OK)

@api\_view(['GET'])

@permission\_classes([IsAuthenticated])

def get\_game(request, game\_id):

    # Получить информацию об одной игре

    if request.method == "GET":

        try:

            game = Game.objects.get(id=game\_id)

            return JsonResponse({

                "id": game.id,

                "name": game.name,

                "code": game.code,

                "genre": game.genre,

                "picture\_url": game.pictures

            })

        except Game.DoesNotExist:

            return JsonResponse({"error": "Game not found"}, status=404)

@api\_view(['POST'])

@permission\_classes([IsSuperuser])

def add\_game(request):

# Добавить новую игру

if request.method == "POST":

data = json.loads(request.body)

try:

id = data["id"]

name = data["name"]

code\_url = data["code"]

genre = data["genre"]

picture\_url = data["pictures"]

except KeyError:

return JsonResponse({"error": "Missing required fields", "data": data}, status=400)

game = Game.objects.create(

id=id,

name=name,

code=code\_url,

genre=genre,

pictures=picture\_url

)

return JsonResponse({

"message": "Game created",

"game": {

"id": game.id,

"name": game.name,

"code": game.code,

"genre": game.genre,

"pictures": game.pictures

}

})